# Application of Forth CPU for control and debugging of FPGA-implemented systems

Wojciech M. Zabołotny[1], Grzegorz H. Kasprowicz[1],

[1]Institute of Electronic Systems, Warsaw University of Technology

XLIV-th IEEE-SPIE Joint Symposium Wilga 2019

# Initialization and debugging of FPGA-based systems


- The HDL description of FPGA-based system allows to assign initial values to the internal signals

# Initialization and debugging of FPGA-based systems


- The HDL description of FPGA-based system allows to assign initial values to the internal signals
- However, sometimes more complex initialization procedure may be needed

# Initialization and debugging of FPGA-based systems


- The HDL description of FPGA-based system allows to assign initial values to the internal signals
- However, sometimes more complex initialization procedure may be needed
- In simple cases it may provided by small state machines

# Initialization and debugging of FPGA-based systems


- The HDL description of FPGA-based system allows to assign initial values to the internal signals
- However, sometimes more complex initialization procedure may be needed
- In simple cases it may provided by small state machines
- In more complicated cases initialization via control bus may be needed

# Initialization and debugging of FPGA-based systems


- The HDL description of FPGA-based system allows to assign initial values to the internal signals
- However, sometimes more complex initialization procedure may be needed
- In simple cases it may provided by small state machines
- In more complicated cases initialization via control bus may be needed
- Similarly debugging and testing of the system usually requires access via control bus

# Initialization and debugging of FPGA-based systems

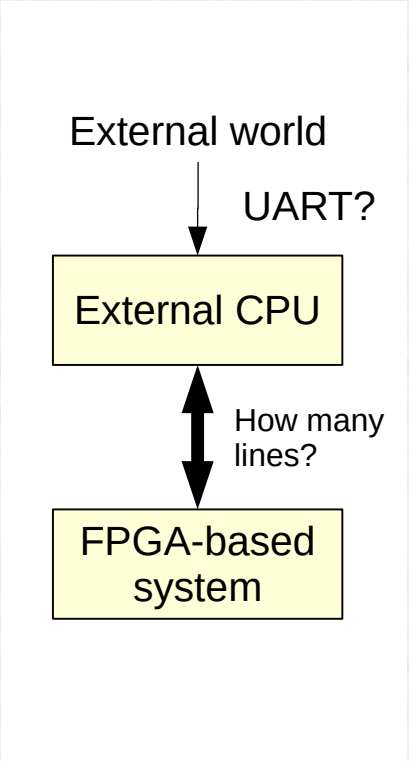

- The HDL description of FPGA-based system allows to assign initial values to the internal signals
- However, sometimes more complex initialization procedure may be needed
- In simple cases it may provided by small state machines
- In more complicated cases initialization via control bus may be needed
- Similarly debugging and testing of the system usually requires access via control bus
- Those tasks may be done from external computer via control interface, e.g., IPbus

# Initialization and debugging of FPGA-based systems


- The HDL description of FPGA-based system allows to assign initial values to the internal signals
- However, sometimes more complex initialization procedure may be needed
- In simple cases it may provided by small state machines
- In more complicated cases initialization via control bus may be needed
- Similarly debugging and testing of the system usually requires access via control bus
- Those tasks may be done from external computer via control interface, e.g., IPbus
- However, in case if we need an autonomous initialization of our system, the local CPU may be needed

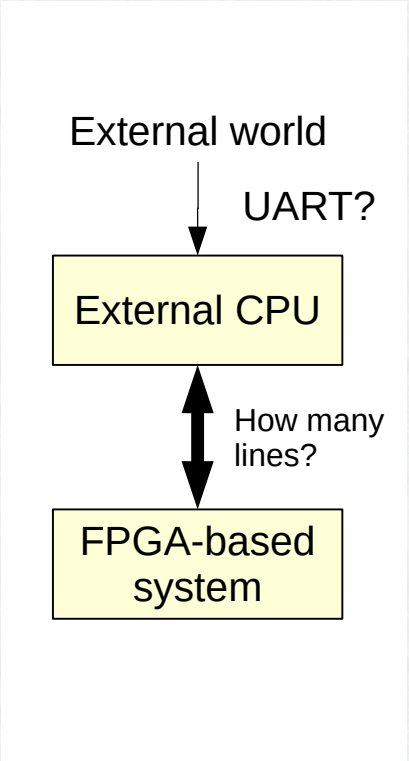
# Local CPU

- The easiest method to solve that problem is to connect an external CPU (e.g., a simple and cheap ARM)



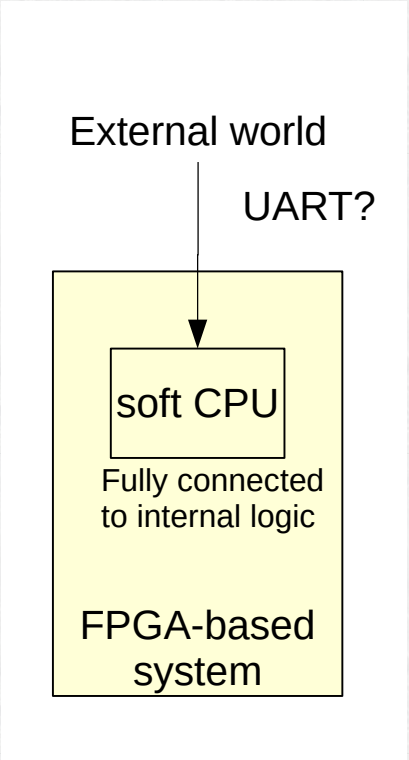
# Local CPU

- The easiest method to solve that problem is to connect an external CPU (e.g., a simple and cheap ARM)
- That must be done at the PCB design stage, and increases complexity of the board



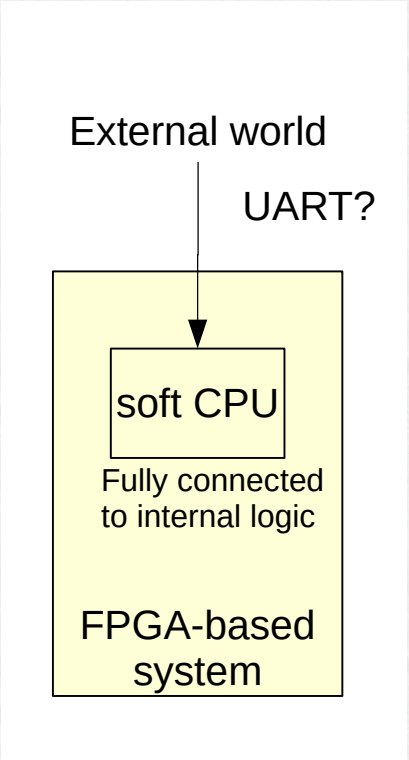
# Local CPU

- The easiest method to solve that problem is to connect an external CPU (e.g., a simple and cheap ARM)
- That must be done at the PCB design stage, and increases complexity of the board
- Another possibility is to use a soft CPU implemented in FPGA



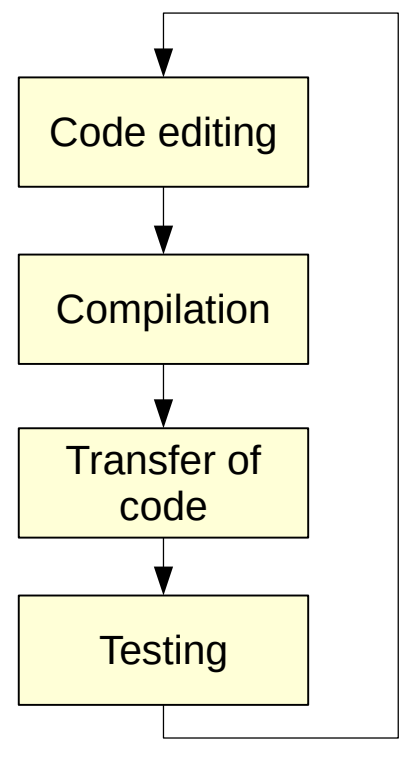
# Local CPU

- The easiest method to solve that problem is to connect an external CPU (e.g., a simple and cheap ARM)
- That must be done at the PCB design stage, and increases complexity of the board
- Another possibility is to use a soft CPU implemented in FPGA
- There are many soft CPUs that may be programmed in C - LatticeMico32, microblaze, RiscV and many more...



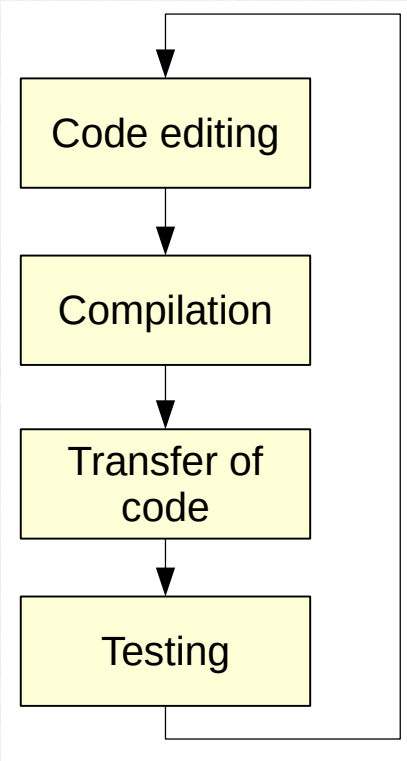
# Local CPU

- The easiest method to solve that problem is to connect an external CPU (e.g., a simple and cheap ARM)
- That must be done at the PCB design stage, and increases complexity of the board
- Another possibility is to use a soft CPU implemented in FPGA
- There are many soft CPUs that may be programmed in C - LatticeMico32, microblaze, RiscV and many more...
- Modification of the program is long (requires editing of source code and recompilation), and the dedicated C toolchain must be available



# Local CPU

- The easiest method to solve that problem is to connect an external CPU (e.g., a simple and cheap ARM)
- That must be done at the PCB design stage, and increases complexity of the board
- Another possibility is to use a soft CPU implemented in FPGA
- There are many soft CPUs that may be programmed in C - LatticeMico32, microblaze, RiscV and many more...
- Modification of the program is long (requires editing of source code and recompilation), and the dedicated C toolchain must be available
- Another possibility is to use Forth-capable CPU



# Forth language


- Language created for embedded and real-time applications [1]

# Forth language


- Language created for embedded and real-time applications [1]
- Very efficient for interactive work

# Forth language


- Language created for embedded and real-time applications [1]
- Very efficient for interactive work
- Typically development is done using the bottom-up approach

# Forth language


- Language created for embedded and real-time applications [1]
- Very efficient for interactive work
- Typically development is done using the bottom-up approach
- Complex routines (words) are created from well tested simple lower level words

# Forth language


- Language created for embedded and real-time applications [1]
- Very efficient for interactive work
- Typically development is done using the bottom-up approach
- Complex routines (words) are created from well tested simple lower level words
- Forth offers compact code with reasonable speed od execution

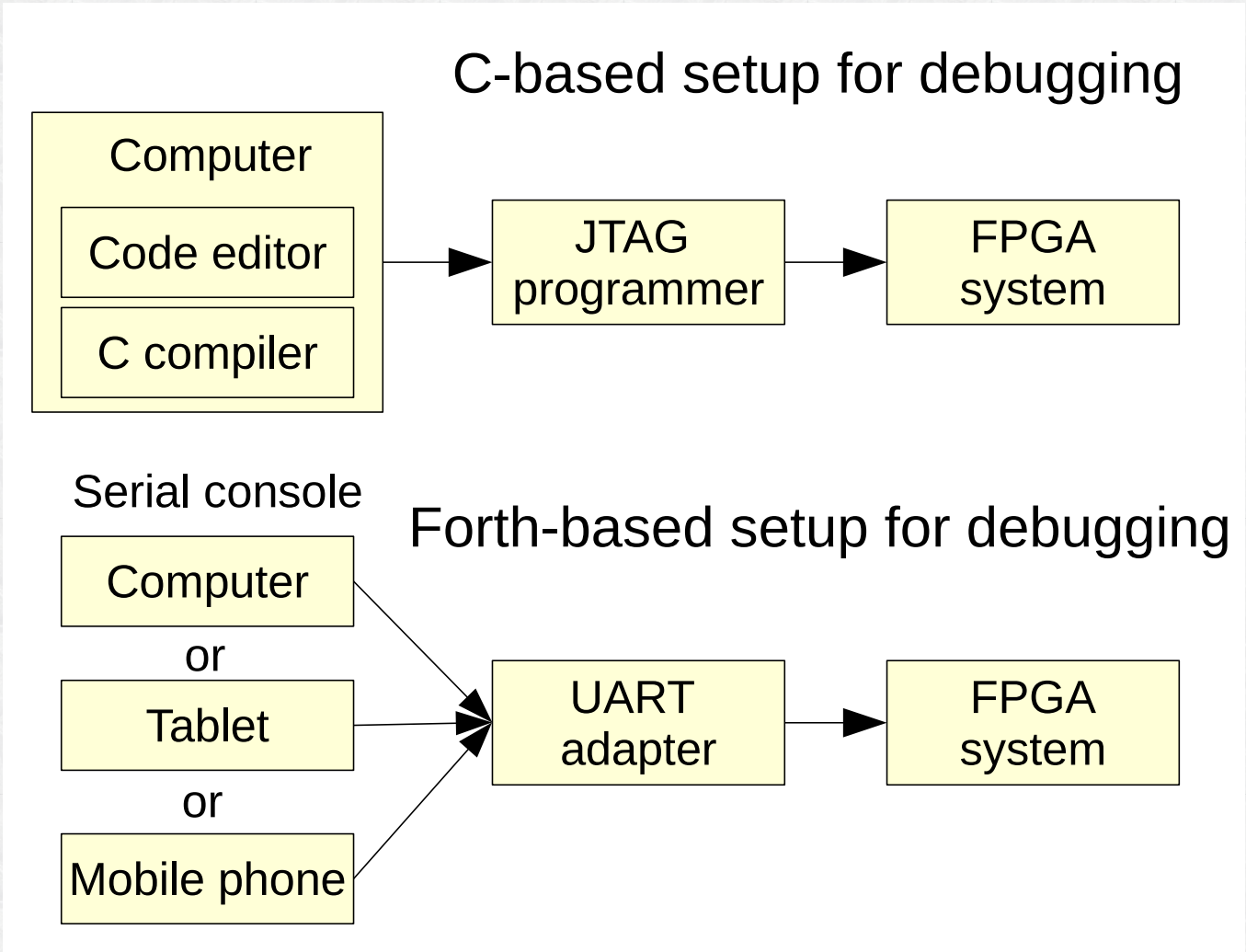
# Forth language


- Language created for embedded and real-time applications [1]
- Very efficient for interactive work
- Typically development is done using the bottom-up approach
- Complex routines (words) are created from well tested simple lower level words
- Forth offers compact code with reasonable speed od execution
- It is possible to create complex, modular applications via incremental compilation

# Forth language


- Language created for embedded and real-time applications [1]
- Very efficient for interactive work
- Typically development is done using the bottom-up approach
- Complex routines (words) are created from well tested simple lower level words
- Forth offers compact code with reasonable speed od execution
- It is possible to create complex, modular applications via incremental compilation
- The whole compiler and Forth CPU may be implemented in simple hardware, all what is needed for development is a serial console

# Comparison of required hardware



# Typical work with Forth


- Use defined words interactively

```
>2 3 * .
 6  ok
>
```

# Typical work with Forth


- Use defined words interactively
- Create new words with usefull sequences of existing words

```
>: fac ( n -- n! )
   1 swap 1+ 1
   ?do i *
   loop ;
   ok
>
```

# Typical work with Forth


- Use defined words interactively
- Create new words with usefull sequences of existing words
- Use the new words together with the previous ones

```
> 2 3 + fac
  ok
>.
 120  ok
>
```

# Typical work with Forth


- Use defined words interactively
- Create new words with usefull sequences of existing words
- Use the new words together with the previous ones
- How to avoid filling memory with incorrect definitions?

```
> 2 3 + fac
  ok
>.
 120  ok
>
```

# Typical work with Forth


- Use defined words interactively
- Create new words with usefull sequences of existing words
- Use the new words together with the previous ones
- How to avoid filling memory with incorrect definitions?
- It is possible to save the state of the system using the *marker* word, and restore it later

```
> 2 3 + fac
  ok
>.
 120  ok
>
```

# Interactive work and creating of programs


- How to create the program when working in interactive mode?

# Interactive work and creating of programs


- How to create the program when working in interactive mode?
- We may simply dump the code memory contents at the end of the session and load it at the begining of th e new one. But what if we want sources?

# Interactive work and creating of programs


- How to create the program when working in interactive mode?
- We may simply dump the code memory contents at the end of the session and load it at the begining of th e new one. But what if we want sources?
- If sources are not available, then we have “write only language”...

# Interactive work and creating of programs


- How to create the program when working in interactive mode?
- We may simply dump the code memory contents at the end of the session and load it at the begining of th e new one. But what if we want sources?
- If sources are not available, then we have “write only language”...
- We can capture our commands in the terminal program.

# Interactive work and creating of programs


- How to create the program when working in interactive mode?
- We may simply dump the code memory contents at the end of the session and load it at the begining of th e new one. But what if we want sources?
- If sources are not available, then we have “write only language”...
- We can capture our commands in the terminal program.
- The captured definitions may be then moved to the source files.

# Interactive work and creating of programs


- How to create the program when working in interactive mode?
- We may simply dump the code memory contents at the end of the session and load it at the begining of th e new one. But what if we want sources?
- If sources are not available, then we have “write only language”...
- We can capture our commands in the terminal program.
- The captured definitions may be then moved to the source files.
- At the begining of the new session we may transfer those files to the Forth CPU

# Examples of Forth code


- Stack based language
- Reverse Polish notation used in calculations
- Limited support for local variables

```
\ i2c_wr1 writes a single byte
: i2c_wr1 ( dta addr -- )
    2* i2c_slv
    I2C_REGS 3 + io!
    64 16 or
    I2C_REGS 4 + io!
    begin
        I2C_REGS 4 + io@
        dup 2 and
    while
            drop
    repeat
  128 and if
      \ NACK in data
      134 err_halt
  then
;
```

# Forth for FPGA


- There are multiple implementations of Forth CPU in HDL (Verilog or VHDL)
  - http://www.forth.org/cores.html
  - http://www.ultratechnology.com/chips.htm
- We have tried to implement our own “tethered” version [2].
- The most successful implementation seems to be the J1 CPU designed by James Bowman [3].
- The original version is implemented in Verilog in 117 lines [4].

# J1B based Forth


- The Forth CPU is only one component of the successful Forth system
- What's needed is also the Forth compiler/interpreter with libraries

# J1B based Forth


- The Forth CPU is only one component of the successful Forth system
- What's needed is also the Forth compiler/interpreter with libraries
- The Swapforth has been prepared for J1B and for other platforms

# J1B based Forth

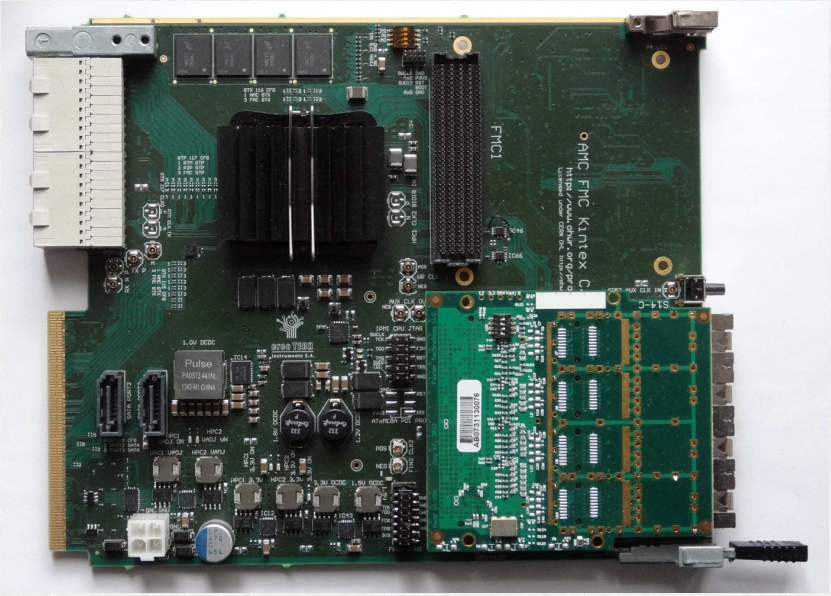

- The Forth CPU is only one component of the successful Forth system
- What's needed is also the Forth compiler/interpreter with libraries
- The Swapforth has been prepared for J1B and for other platforms
- It is supplemented with convenient shell written in Python that supports:
  - saving the commands to the history file
  - dumping the memory contents to the file
  - loading source files to the Forth CPU

# J1B based Forth


- The Forth CPU is only one component of the successful Forth system
- What's needed is also the Forth compiler/interpreter with libraries
- The Swapforth has been prepared for J1B and for other platforms
- It is supplemented with convenient shell written in Python that supports:
  - saving the commands to the history file
  - dumping the memory contents to the file
  - loading source files to the Forth CPU
- The "cold" word, if defined, is executed after the powerup or reset

# AFCK board controller

- Project developed for AFCK boards used as DPB prototype in CBM experiment
- Forth CPU uses I2C interface
  - to read the MAC address
  - to configure the Silabs Si57x clock generator
  - to configure clock switch matrix
- Support for Si57x required implementation of multiple precision arithmetics library
- The status of the initialization routine may be stored in register available for the firmware

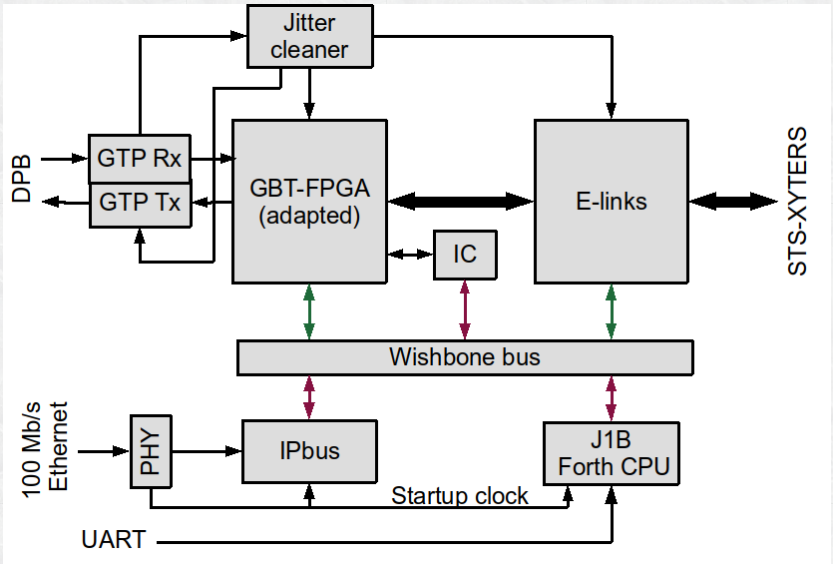


# Example word from arithmetics library


```
\ Definition of the unsigned double compare
: ud< ( a0 a1 b0 b1 -- flag )
    \ If MSW are equal, check the LSW
   rot ( a0 b0 b1 a1 )
   over over = if ( a0 b0 b1 a1 )
       \ MSWs are equal, so compare LSWs
      drop drop ( a0 b0 )
      u<
   else ( a0 b0 b1 a1 )
       \ MSWs are not equal, so their comparison produces the
      u> >r ( a0 b0 )
      drop drop r>
   then
;
```

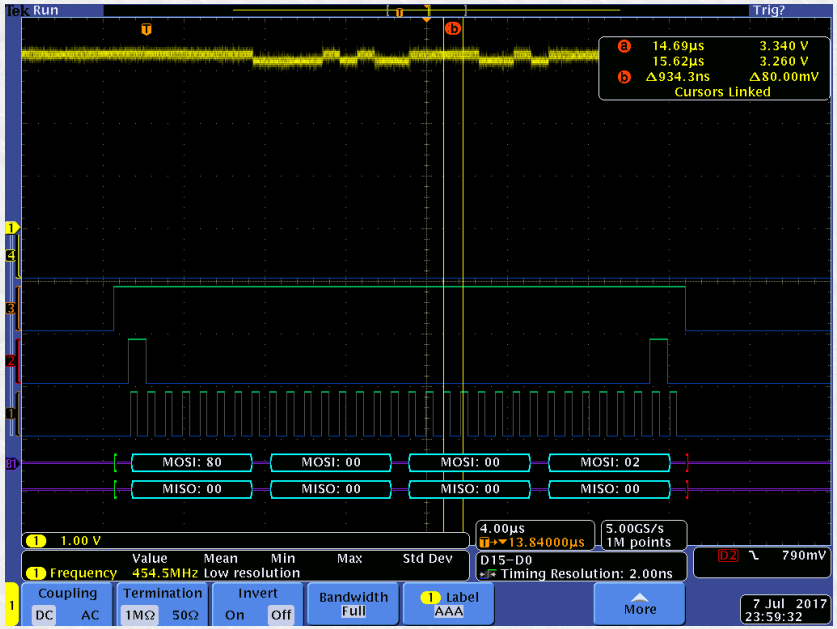
# GBTxEMU System controller

- Forth CPU is one of three masters of the internal Wishbone bus
- Its task is to initialize the system at powerup so that it can be further controlled via optical link
- Later on it can be used for interactive debugging and testing
- Access to internal registers is supported by address tables generated by the addr_gen_wb framework.



# Sayma board controller

- Library developed for Sinara project
- Implements support for many SPI connected peripherals



# How to start with Forth


- Experimenting with Forth does not require FPGA board
- J1B provides also the emulated environment based on Verilator
- Gforth is available for PC
- There are many implementations for microcontrollers. Just a few examples:
  - Mecrisp for ARMs
  - FlashForth for AVR and PIC microcontrollers
  - Amforth for AVR and RISC-V
  - Punyforth for ESP8266

# Conclusions


- Forth based CPU may be a convenient tool for initialization and interactive debugging of FPGA-based systems
- It allows interactive debugging and testing of hardware

# Conclusions


- Forth based CPU may be a convenient tool for initialization and interactive debugging of FPGA-based systems
- It allows interactive debugging and testing of hardware
- Words created in interactive session may be used to create complex applications, e.g. used for initialization of the hardware.
- The ready application may be automatically started after power-up

# Conclusions


- Forth based CPU may be a convenient tool for initialization and interactive debugging of FPGA-based systems
- It allows interactive debugging and testing of hardware
- Words created in interactive session may be used to create complex applications, e.g. used for initialization of the hardware.
- The ready application may be automatically started after power-up
- Forth may be a good solution for interactive in-field debugging or testing also of MCU-based systems

# Bibliography


[1] Leo Brodie.
Thinking Forth.
http://thinking-forth.sourceforge.net/ .

[2] Paweł Goździkowski and Wojciech M. Zabołotny.
Tethered Forth system for FPGA applications.
*Proc. SPIE*, 8903:89031M, October 2013.

[3] James Bowman.
The J1 forth CPU, 2010.
https://www.excamera.com/sphinx/fpga-j1.html .

[4] James Bowman.
The J1B source code, 2010.
https://github.com/jamesbowman/swapforth/blob/master/j1b/verilog/j1.v .